Алгоритмы: сортировки | Я.Шпора

Устойчивость сортировок

Если после применения сортировки первоначальный порядок элементов с равными значениями сохраняется, такая сортировка называется **устойчивой** или **стабильной** (англ. *stable sort*). Если в ходе сортировки элементы с равным значением меняются местами — сортировка неустойчива.

Сортировка вставками

При сортировке вставками элементы массива переставляют по одному так, чтобы после каждой вставки левая часть массива оставалась отсортированной.

Пример реализации алгоритма сортировки вставками:

```
def insertion_sort(arr):
    # Проходим по всем элементам массива, начиная со второго.
    for i in range(1, len(arr)):
        # Сохраняем текущий элемент в переменную temp.
        temp = arr[i]
        # Сохраняем индекс предыдущего элемента в переменную j.
        j = i - 1
        # Сравниваем temp с предыдущим элементом
        # и двигаем предыдущий элемент на одну позицию вправо,
        # пока он больше temp и не достигнул начала массива.
        while j >= 0 and arr[j] > temp:
            arr[j + 1] = arr[j]
            j -= 1
        # Вставляем temp в отсортированную часть массива на нужное место.
        arr[j + 1] = temp
        print(f'Шаг {i}, отсортировано элементов: {i + 1}, {arr}')


insertion_sort([2, 9, 11, 7, 1])
```

Временная сложность алгоритма `O(n^2)` .

Сортировка по ключу

Признак, по которому элементы сравниваются и сортируются, называют **ключом сортировки**.

Обычно функция сортировки принимает в качестве вспомогательного аргумента функцию, описывающую признак, по которому должны быть отсортированы элементы. Есть два распространённых способа это сделать.

Первый вариант: в алгоритм сортировки в качестве параметра передаётся лямбда-функция или обычная функция с одним параметром.

```
words = ['слово', 'дом', 'электрификация', 'абракадабра']


# Применим сортировку вставками.
def insertion_sort_by_key(arr, key):
    for i in range(1, len(arr)):
        temp = arr[i]
        j = i - 1
        # При сравнении вызываем функцию, возвращающую значение ключа.
        while j >= 0 and key(arr[j]) > key(temp):
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = temp



# При вызове сортировки передаем в параметры функцию,
# возвращающую длину слова.
insertion_sort_by_key(words, lambda x: len(x))
print(words)

# При вызове сортировки передаем в параметры функцию,
# возвращающую само слово. Элементы будут отсортированы по алфавиту.
insertion_sort_by_key(words, lambda x: x)
print(words)

# Вывод
# ['дом', 'слово', 'абракадабра', 'электрификация']
```

```
# ['абракадабра', 'дом', 'слово', 'электрификация']
```

Второй вариант: в алгоритм сортировки передаётся функция с двумя аргументами: `is_first_word_shorter(str1, str2)` . Эта функция сравнивает два объекта и возвращает `True` , если первый объект должен быть расположен раньше второго, и `False` в противном случае.

```
# Отсортируем список по длине первого слова в элементе.
words = [
    'слово дня',
    'дом солнца',
    'электрификация страны',
    'абракадабра в статье'
]


# Функция-компаратор, сравнивающая длину первых слов в строке:
def is_first_word_shorter(str1, str2):
    # Получаем первые слова из строк
    word1 = str1.split()[0]
    word2 = str2.split()[0]
    return len(word1) < len(word2)


# Применим сортировку вставками.
def insertion_sort_by_comparator(arr, less):
    for i in range(1, len(arr)):
        temp = arr[i]
        j = i - 1
        while j >= 0 and less(temp, arr[j]):
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = temp


insertion_sort_by_comparator(words, is_first_word_shorter)
print(words)
```

```
    # Вывод
    # ['дом солнца', 'слово дня', 'абракадабра в статье', 'электрификация
```

Сортировка слиянием

Принцип работы сортировки слиянием:

1. Массив разбивается на две части, пополам.
2. Если в каком-то из получившихся подмассивов больше одного элемента, то для него рекурсивно запускается разделение на части: см. п.1.
3. Когда в подмассивах после разделения на части остаётся по одному элементу, два подмассива соединяются в один. Получившиеся массивы объединяются со своими половинами — и так до полной сборки массива.

Реализация алгоритма выглядит так:

```
def merge_sort(array):
    # Базовый случай рекурсии.
    if len(array) <= 1:
        return array

    # Рекурсивный разбор массива в левой половине:
    # передаём в merge_sort() левую половину полученного на вход массива.
    left = merge_sort(array[0 : len(array)//2])

    # Рекурсивный разбор массива в правой половине:
    # передаём в merge_sort() правую половину полученного на вход массива
    right = merge_sort(array[len(array)//2 : len(array)])

    return merge(left, right)


# Функция сортировки и слияния:
def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0

    while left_idx < len(left) and right_idx < len(right):
        # Сравниваем:
```

```
        if left[left_idx] <= right[right_idx]:
            # Добавляем в result:
            result.append(left[left_idx])
            # Сдвигаем указатель:
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    return result + left[left_idx:] + right[right_idx:]


test_array = [5, 4, 9, 10, 8, 3, 11, 1, 7, 6, 2]
print(merge_sort(test_array))
```

Временная сложность этого алгоритма — `O(n log n)` .

Быстрая сортировка

В быстрой сортировке применяется стратегия «разделяй и властвуй»:

- разделить массив на части меньшего размера;
- рекурсивно вызвать алгоритм сортировки для этих частей;
- объединить результаты, полученные для частей, в общий результат.

Реализация алгоритма:

```
def quicksort(array):
    """Быстрая сортировка."""
    # Базовый случай рекурсии.
    if len(array) <= 1:
        return array
    # Определяем индекс опорного элемента.
    middle_element_index = len(array) // 2
    # Получаем опорный элемент:
    pivot = array[middle_element_index]
    # Передаём в функцию partition() массив и опорный элемент.
    left, center, right = partition(array, pivot)
    # Рекурсивно вызываем quicksort() для левого и правого списков,
```

```
    # а затем соединяем все три списка в один.
    return quicksort(left) + center + quicksort(right)


def partition(array, pivot):
    """
    Разбивает массив на три разных массива относительно опорного элемента
    """
    # Создаём три пустых списка.
    left, center, right = [], [], []
    # Раскладываем элементы по спискам.
    for item in array:
        if item < pivot:
            left.append(item)
        elif item > pivot:
            right.append(item)
        elif item == pivot:
            center.append(item)
    # Возвращаем кортеж с тремя списками.
    return left, center, right


arr = [44, 60, 10, 61, 60, 2, 62, 18, 2, 69]
result = quicksort(arr)
print(result)
```

В среднем случае алгоритм быстрой сортировки выполняется за `O(n log n)`.

Сортировка подсчётом

Этапы сортировки:

1. Считаем, сколько раз каждое уникальное значение встречается в исходном массиве.
2. Формируем итоговый массив, умножая уникальное значение на то количество раз, сколько оно встречается в исходном массиве.

Реализация в коде:

```
from collections import defaultdict
```

```
def counting_sort(array, maximum):
    count = defaultdict(int)
    # Перебираем массив по элементам.
    for item in array:
        # Увеличиваем счётчик появления данного элемента на единицу.
        count[item] += 1

    sorted_array = []
    # Перебираем все уникальные элементы.
    for item in range(maximum + 1):
        # Добавляем к результату уникальный элемент столько раз,
        # сколько он встретился в коде.
        sorted_array += [item] * count[item]
    return sorted_array


arr = [5, 6, 1, 8, 8, 7, 1, 0, 4, 8, 3, 2, 1, 3, 4, 6, 1, 1]
result = counting_sort(arr, 8)
print(result)
# [0, 1, 1, 1, 1, 1, 2, 3, 3, 4, 4, 5, 6, 6, 7, 8, 8, 8]
```

Количество элементов в массиве обозначаем через `n`, а количество уникальных элементов — `r`. Тогда сложность алгоритма будет `O(n + r)`. В оценке сложности этого алгоритма важно учитывать оба параметра.

В мире, полном информации,
шпаргалка — это компас,
указывающий направление
мудрому искателю.

Фернан Мореплаватель